Contents

Ingress Gateways

7 minute read ✓ page test

Along with support for Kubernetes Ingress, Istio offers another configuration model, Istio Gateway. A `Gateway` provides more extensive customization and flexibility than `Ingress`, and allows Istio features such as monitoring and route rules to be applied to traffic entering the cluster.

This task describes how to configure Istio to expose a service outside of the service mesh using an Istio `Gateway`.

Before you begin

- Setup Istio by following the instructions in the Installation guide.
- Make sure your current directory is the `istio` directory.
- Start the httpbin sample.

  If you have enabled automatic sidecar injection, deploy the `httpbin` service:

  ```
  $ kubectl apply -f @samples/httpbin/httpbin.yam
  l@
  ```

  Otherwise, you have to manually inject the sidecar before deploying the `httpbin` application:

  ```
  $ kubectl apply -f <(istioctl kube-inject -f @s
  amples/httpbin/httpbin.yaml@)
  ```

- Determine the ingress IP and ports as

described in the following subsection.

Determining the ingress IP and ports

Execute the following command to determine if your Kubernetes cluster is running in an environment that supports external load balancers:

```
$ kubectl get svc istio-ingressgateway -n istio-sys
tem
NAME                   TYPE           CLUSTER-IP
    EXTERNAL-IP      PORT(S)   AGE
istio-ingressgateway   LoadBalancer   172.21.109.12
9   130.211.10.121   ...       17h
```

If the `EXTERNAL-IP` value is set, your environment has an external load balancer that you can use for the ingress gateway. If the `EXTERNAL-IP` value is `<none>` (or perpetually `<pending>`), your environment does not provide an external load balancer for the ingress gateway. In this case, you can access the gateway using the service's node port.

Choose the instructions corresponding to your environment:

external load balancer node port

Follow these instructions if you have determined that your environment has an external load balancer.

Set the ingress IP and ports:

```
$ export INGRESS_HOST=$(kubectl -n istio-sys
tem get service istio-ingressgateway -o json
path='{.status.loadBalancer.ingress[0].ip}')
$ export INGRESS_PORT=$(kubectl -n istio-sys
tem get service istio-ingressgateway -o json
path='{.spec.ports[?(@.name=="http2")].port}
')
$ export SECURE_INGRESS_PORT=$(kubectl -n is
tio-system get service istio-ingressgateway
-o jsonpath='{.spec.ports[?(@.name=="https")
].port}')
$ export TCP_INGRESS_PORT=$(kubectl -n istio
-system get service istio-ingressgateway -o
jsonpath='{.spec.ports[?(@.name=="tcp")].por
t}')
```

In certain environments, the load balancer may be exposed using a host name, instead of an IP address. In this case, the ingress gateway’s `EXTERNAL-IP` value will not be an IP address, but rather a host name, and the above command will have failed to set the `INGRESS_HOST` environment variable. Use the following command to correct the `INGRESS_HOST` value:

```
$ export INGRESS_HOST=$(kubectl -
n istio-system get service istio-
ingressgateway -o jsonpath='{.sta
tus.loadBalancer.ingress[0].hostn
ame}')
```

Configuring ingress using an Istio gateway

An ingress Gateway describes a load balancer operating at the edge of the mesh that receives incoming HTTP/TCP connections. It configures exposed ports, protocols, etc. but, unlike Kubernetes Ingress Resources, does not include any traffic routing configuration. Traffic routing for ingress traffic is instead configured using Istio routing rules, exactly in the same way as for internal service requests.

Let's see how you can configure a `Gateway` on port 80 for HTTP traffic.

1. Create an Istio `Gateway`:

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: httpbin-gateway
spec:
  selector:
    istio: ingressgateway # use Istio default g
ateway implementation
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "httpbin.example.com"
EOF
```

2. Configure routes for traffic entering via the `Gateway`:

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: httpbin
spec:
  hosts:
  - "httpbin.example.com"
  gateways:
  - httpbin-gateway
  http:
  - match:
    - uri:
        prefix: /status
    - uri:
        prefix: /delay
    route:
    - destination:
        port:
          number: 8000
        host: httpbin
EOF
```

You have now created a virtual service configuration for the `httpbin` service containing two route rules that allow traffic for paths `/status` and `/delay`.

The gateways list specifies that only

requests through your `httpbin-gateway` are allowed. All other external requests will be rejected with a 404 response.

> ⚠ Internal requests from other services in the mesh are not subject to these rules but instead will default to round-robin routing. To apply these rules to internal calls as well, you can add the special value `mesh` to the list of `gateways`. Since the internal hostname for the service is probably different (e.g., `httpbin.default.svc.cluster.local`) from the external one, you will also need to add it to the `hosts` list. Refer to the operations guide for more

details.

3. Access the *httpbin* service using *curl*:

   ```
   $ curl -s -I -HHost:httpbin.example.com "http:/
   /$INGRESS_HOST:$INGRESS_PORT/status/200"
   HTTP/1.1 200 OK
   server: istio-envoy
   ...
   ```

   Note that you use the `-H` flag to set the *Host* HTTP header to “httpbin.example.com”. This is needed because your ingress `Gateway` is configured to handle “httpbin.example.com”, but in your test environment you have no DNS binding for that host and are simply sending your request to the ingress IP.

4. Access any other URL that has not been explicitly exposed. You should see an HTTP 404 error:

```
$ curl -s -I -HHost:httpbin.example.com "http:/
/$INGRESS_HOST:$INGRESS_PORT/headers"
HTTP/1.1 404 Not Found
...
```

Accessing ingress services using a browser

Entering the `httpbin` service URL in a browser won’t work because you can’t pass the *Host* header to a browser like you did with `curl`. In a real world situation, this is not a problem because you configure the requested host properly and DNS resolvable. Thus, you use the host’s domain name in the URL, for example, `https://httpbin.example.com/status/200`.

To work around this problem for simple tests and demos, use a wildcard `*` value for the host in the `Gateway` and `VirtualService` configurations. For example, if you change your ingress configuration to the following:

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: httpbin-gateway
spec:
  selector:
    istio: ingressgateway # use Istio default gatew
ay implementation
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: httpbin
spec:
  hosts:
  - "*"
  gateways:
  - httpbin-gateway
  http:
  - match:
    - uri:
        prefix: /headers
    route:
    - destination:
        port:
          number: 8000
        host: httpbin
EOF
```

You can then use `$INGRESS_HOST:$INGRESS_PORT` in the browser URL. For example, `http://$INGRESS_HOST:$INGRESS_PORT/headers` will display all the headers that your browser sends.

Understanding what happened

The `Gateway` configuration resources allow external traffic to enter the Istio service mesh and make the traffic management and policy features of Istio available for edge services.

In the preceding steps, you created a service inside the service mesh and exposed an HTTP endpoint of the service to external traffic.

Troubleshooting

1. Inspect the values of the `INGRESS_HOST` and `INGRESS_PORT` environment variables. Make sure they have valid values, according to the output of the following commands:

```
$ kubectl get svc -n istio-system
$ echo "INGRESS_HOST=$INGRESS_HOST, INGRESS_POR
T=$INGRESS_PORT"
```

2. Check that you have no other Istio ingress gateways defined on the same port:

   ```
   $ kubectl get gateway --all-namespaces
   ```

3. Check that you have no Kubernetes Ingress resources defined on the same IP and port:

   ```
   $ kubectl get ingress --all-namespaces
   ```

4. If you have an external load balancer and it does not work for you, try to access the gateway using its node port.

Cleanup

Delete the `Gateway` and `VirtualService` configuration, and shutdown the httpbin service:

```
$ kubectl delete gateway httpbin-gateway
$ kubectl delete virtualservice httpbin
$ kubectl delete --ignore-not-found=true -f @sample
s/httpbin/httpbin.yaml@
```